Relatório do Trabalho Prático I: otimização da multiplicação sequencial entre matrizes reais

André Brait Carneiro Fabotti
Rodrigo Rodrigues de Novaes Júnior

Setembro de 2017

1 Introdução

Esse trabalho tem como objetivo otimizar a multiplicação sequencial entre duas matrizes reais $A_{m \times n}$ e $B_{n \times m}$, de acordo com o enunciado presente no Apêndice A. Foram utilizadas três abordagens:

- *naïve*

  Implementa o algoritmo de multiplicação entre matrizes ijk, com $2n^3$ operações em ponto flutuante;

- *blocked*

  Distribuição da multiplicação em blocos, cujo tamanho caiba na linha de entrada na CACHE;

- implementação do algoritmo de Strassen

  Divisão da operação em submatrizes e redução no número de multiplicações realizadas.

Os resultados apresentaram dados consistentes, sendo possível avaliar a quantidade de operações de ponto flutuante por segundo, em Mflop/s, em função da dimensão da matriz, para cada uma das abordagens listadas. De todas, a que alcançou a maior média de percentual dos picos foi a abordagem *naïve*, embora o melhor desempenho geral tenha sido alcançado pela abordagem *blocked*. O algoritmo de Strassen, apesar de implementado corretamente, não obteve resultados satisfatórios.

2 Metodologia

O *benchmark* foi modificado da seguinte forma:

- recebe como argumento um valor inteiro s_{max}, correspondente ao valor máximo da dimensão de uma matriz, cujo vetor de testes é gerado no intervalo $1 \leq 2^k < s_{max}$, $k = 1, 2, \ldots, log_2(s_{max})$. Se s_{max} não for indicado, então serão aplicados os valores-padrão;

- criação do script *execute_ experiments.sh*, que constrói o projeto e executa os experimentos para cada dimensão no vetor de testes. O script também pode receber s_{max} como argumento.

Os experimentos foram executados utilizando o subconjunto padrão, dado no arquivo *benchmark.c*, e para $s_{max} = 10000$, que executa testes para matrizes de dimensão $n = 1, 2, 4, \ldots, 1024, 2048, 4096$. Os experimentos foram executados na plataforma Dell Vostro 5470, 4 GB RAM, Intel© Core© i5-4210U @ 1.70GHz × 4, sistema operacional ubuntu 16.04.3 LTS. Os experimentos não foram realizados na Katrina, pois, conforme no enunciado dado no Apêndice A, o seu uso figurar "somente dentro do CEFET", de modo que para "executar comandos nessa máquina", em outro lugar, deve-se entrar "em contato" com a professora, só tornou de ciência dos alunos recentemente. Por essa falha, pedimos sinceras desculpas e que considere os resultados coletados na plataforma mencionada.

3 Resultados

Nesta Seção, serão mostrados e discutidos os gráficos de desempenho em função da dimensão da matriz, de acordo com cada abordagem dada na Seção 1.

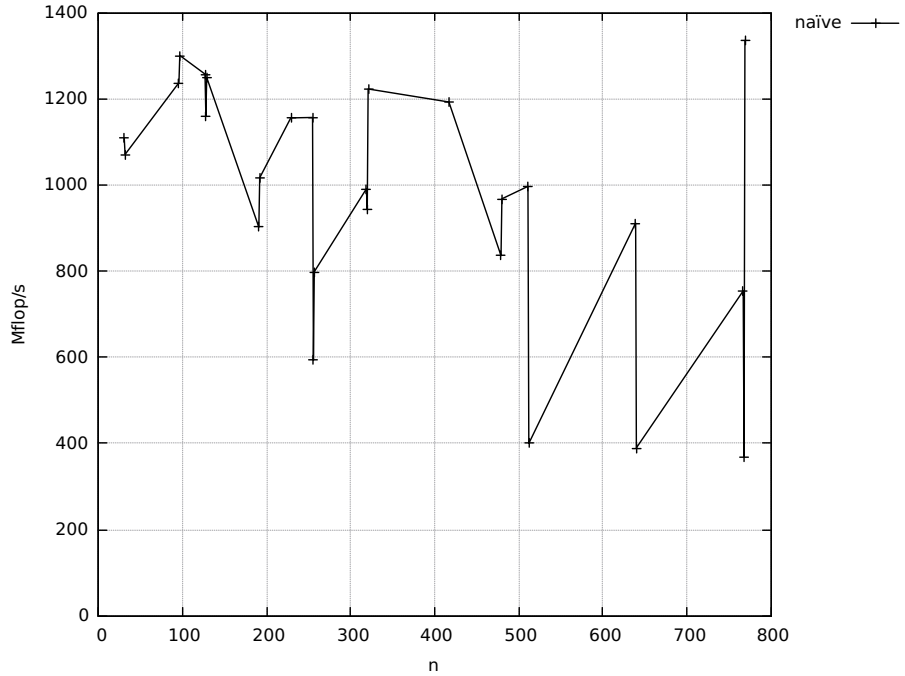

Figura 1: Desempenho da abordagem *naïve* em Mflop/s × n, onde n é a dimensão da matriz.

O gráfico da Figura 1 apresenta os resultados obtidos para a abordagem *naïve*. Nota-se que houve grandes variações à medida em que a dimensão da

matriz aumentava. Além disso, quanto maior a dimensão, menor o desempenho observado. Isso mostra que há um impacto negativo da dimensão da matriz sobre a quantidade de operações em ponto flutuante executada, i.e., há menor aproveitamento do processador à medida em que o tamanho da matriz aumenta. Além disso, apesar de o desempenho máximo ter alcançado cerca de 1400 Mflop/s, para certas entradas foi observado um mínimo de aproximadamente 400 Mflop/s, acentuando a instabilidade dessa abordagem para multiplicação de matrizes.

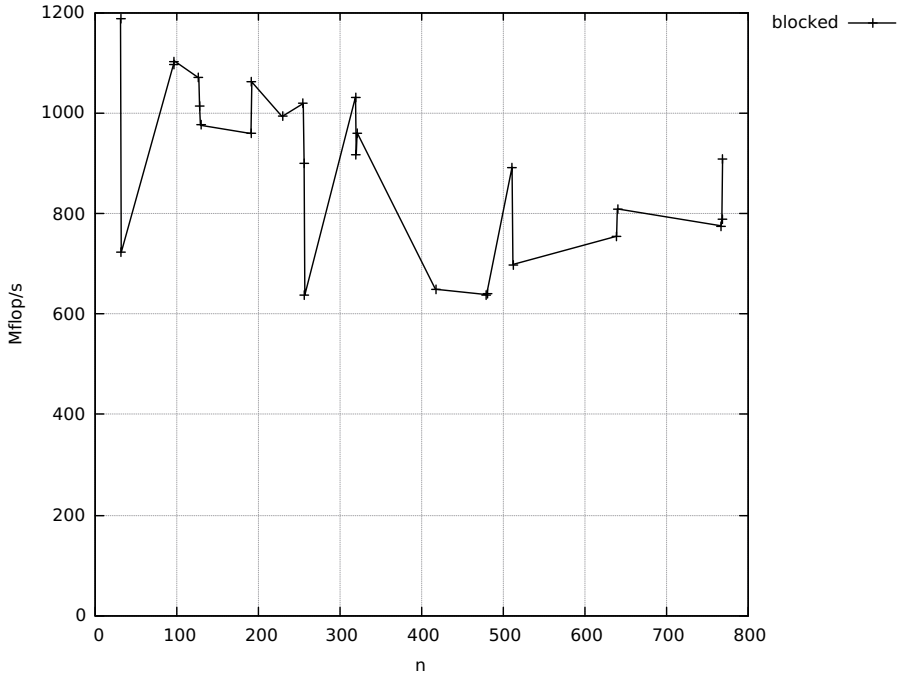

Figura 2: Desempenho da abordagem *blocked* em Mflop/s $\times$ n, onde n é a dimensão da matriz.

Na Figura 2, o gráfico representa o desempenho em função da dimensão da matriz para a abordagem *blocked*. Nesta foram observadas menos variações acentuadas, ainda que um desempenho máximo inferior ao da abordagem *naïve*. No entanto, observa-se que o desempenho mínimo encontrado é melhor que o da abordagem anterior, que juntamente à maior instabilidade e menores variações nítidas de desempenho, proporcionaram um melhor uso geral do processador.

A Figura 3 mostra o desempenho da abordagem *blocked* em função do crescimento exponencial da dimensão matricial. Ora, considerando que o algoritmo de Strassen não conseguiu alocar memória suficiente e que a abordagm *naïve* permaneceu sendo processada por horas, assumiu-se que a *blocked* é aquela que obtém os melhores desempenhos para matrizes muito grandes.

Na Figura 4, observa-se o desempenho obtido na implementação do algoritmo de Strassen. Nota-se, claramente, que foi a abordagem com menos variações acentuadas, embora tenha o pior resultado. Paradoxalmente, é aquele que possui a menor complexidade computacional, uma vez que realiza menos que $O(n^3)$

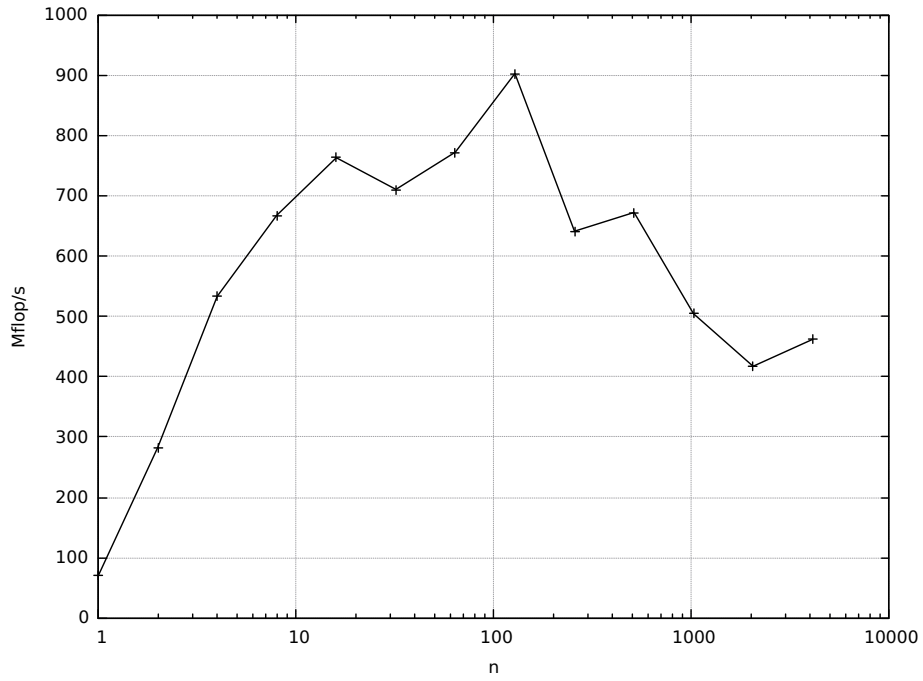

Figura 3: Desempenho da abordagem *blocked* em Mflop/s $\times$ n, onde n é a dimensão da matriz.

operações em pontos flutuantes. Portanto, o desempenho pode ser justificado pelo mal aproveitamento da memória e do processador, pois o algoritmo aloca muitas matrizes dinamicamente e é recursivo. Além disso, sabendo que se trata de um algoritmo de divisão e conquista, há a possibilidade de que o paralelismo possa surtir bons efeitos sobre o desempenho geral do programa.

As questões avaliadas ao longo desta Seção carecem de maiores verificações, já que os experimentos foram realizados em uma plataforma com menor capacidade de memória e processamento. Há, também, as instabilidades que podem surgir devido à execução de processos simultâneos. Por fim, o trabalho foi satisfatório para compreender o quão difícil pode ser otimizar um código sequencial, especialmente no caso de multiplicação matricial, mesmo quando se implementa um algoritmo com menor complexidade assintótica. É preciso ver que, na prática, as coisas nem sempre se comportam da forma esperada.

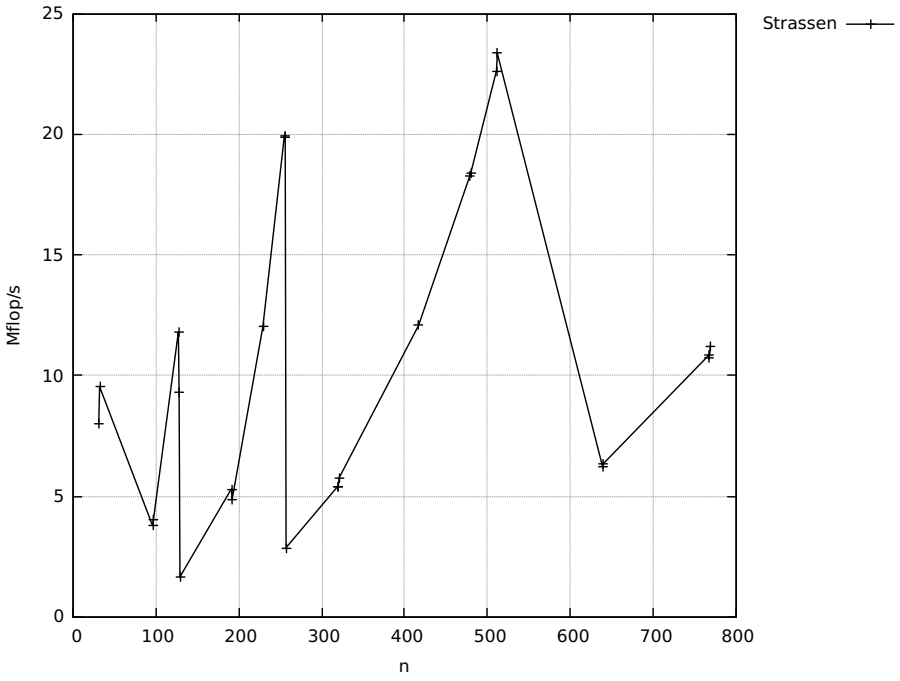

Figura 4: Desempenho da implementação do algoritmo de Strassen em Mflop/s $\times$ n, onde n é a dimensão da matriz.

A Enunciado do Trabalho Prático I

CEFET-MG

TP1: Multiplicação de matrizes

Descrição do problema:

Nesse trabalho você deve otimizar a **Multiplicação de matrizes** (**sequencial**).

Considere que as matrizes A e B de entrada são matrizes quadradas nxn de doubles (https://en.wikipedia.org/wiki/Double-precision_floating-point_format). Isto pode ser implementado usando $2n^3$ operações em ponto flutuante (n^3 somas, n^3 multiplicações), como no seguinte pseudocódigo::

```
for i = 1 to n
  for j = 1 to n
    for k = 1 to n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    end
  end
end
```

Instruções

- O trabalho pode ser feito em dupla. Não copie códigos prontos da internet, estamos aqui para aprender.
- **O código TEM que estar correto, códigos retornando resultados errados serão anulados.** Em um código correto:

```
|square_dgemm(n,A,B,0) - A*B| < eps*n*|A|*|B|.
```

onde `eps` := 2^{-52} = 2.2 * 10^{-16} (eps= machine epsilon). (Veja o arquivo benchmark.c: Programa para medir o tempo e verificar a correção do algoritmo)

- A submissão deve ser feita em um tar.gz com nome matmult_integrantes_do_grupo.tar.gz através do link disponibilizado no Moodle (ava.cefetmg.br), contendo:

* dgemm-blocked.c o código fonte C que contém a sua implementação da rotina:

```
  void square_dgemm(int, double*, double*, double*);
```

descrita no pseudocódigo acima. Veja no arquivo em anexo uma implementação de exemplo.
* Relatorio.nome.pdf, usar latex: \documentclass[a4paper,10pt]{article}), contendo:

- Introdução: Descreva brevemente qual o objetivo do trabalho da equipe
- Metodologia: Explique por extenso quais métodos usou, o que deu errado, o que deu certo, resultados em cada caso.
- Resultados computacionais: Ambiente computacional, planejamento experimental (objetivo do experimento e como vai ser executado: ex. para comparar o desempenho vou medir o tempo de execução

com tais valores de N para diferentes métodos), como mudou o desempenho comparado com sua máquina (desktop/laptop), Figuras com gráficos (usando o gnuplot) de tamanho da matriz vs tempo de execução. Discussão de cada resultado. Documente cada experimento que executar ainda que ele não tenha melhorado o tempo de execução e justifique o que aconteceu. Você deverá executar seus testes com matrizes de diferentes tamanhos. Ex.: 400, 1600, 3600, 6400, 10000. Compare o resultado com os valores obtidos compilando com

```
-O3 -fstrict-aliasing -std=c99
```

(Sobre a keyword C99 `restrict` e pointer-aliasing, veja este artigo.)

- Conclusões
- Seu código usa precisão dupla para representar números reais.Uma referência comum para a multiplicação de matrizes com componentes reais é a rotina `dgemm` (double-precision general matrix-matrix multiply) no nível 3 BLAS. Como parte da avaliação iremos comparar a sua implementação com a dgemm otimizada oferecida na biblioteca BLAS (veja o arquivo benchmark.c).
- A **nota** irá depender do resultado dessa comparação e **da qualidade do seu texto**.
- Você pode usar o compilador de sua preferência, sem otimizações. No caso do GNU C compiler (`gcc`), compile usando gcc -O0. Nesse trabalho não usaremos vetorização.
- As matrizes estão armazenadas **coluna a coluna** ou seja, $C_{i,j}$ == `C(i,j)` == `C[(i-1)+(j-1)*n]`, for `i`=1:n, onde n é o número de linhas em `C (usamos indexação 1 para símbolos matemáticos e notação de índices em MATLAB (entre parêntesis) e` indexação baseada em 0 quando usamos a notação de índices C (A[i,j]).Se o seu algoritmo indexar a matriz seguindo a ordem das linhas vai ficar **MUITO** ruim. (A biblioteca é feita em Fortran)
- Seu código só pode chamar funções da biblioteca standard C.
- O processador alvo é um Intel Haswell-EP de 6 cores (12 threads) executando a 2.4 GHz, resultando em:

8 double-precision (64-bit) flops por pipeline * 6 pipelines * 2.4 GHz = 115.2 Gflops/s. (Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40GHz, (CPU speed in GHz) x (number of CPU cores) x (CPU instruction per cycle) x (number of CPUs per node).)

- Você pode implementar algum algoritmo com complexidade assintótica menor que O(n^3) (ex. Strassen https://en.wikipedia.org/wiki/Strassen_algorithm) valendo um ponto adicional, mas ele não será considerado na comparação de desempenho entre algoritmos sequenciais.
- Você precisará uma conta na katrina (200.131.37.129), entre em contato para criar sua conta. O comando ssh:

ssh seulogin@200.131.37.222 -p 2200

funciona somente dentro do CEFET. Para executar comandos nessa máquina desde fora do CEFET, entre em contato. Para dúvidas e discussões, use o Fórum da disciplina.

- Não deixe os testes para o último dia. Coisas misteriosas podem acontecer com o servidor.

Dicas:

- Experimente com vários tamanhos de bloco e vários níveis de cache;
- Melhore seu código de forma incremental;
- Teste desenrolar laços e otimizações de cópia, troque os laços de ordem, etc..

Guias de uso do gnuplot:
http://www.uft.edu.br/engambiental/prof/catalunha/arquivos/gnuplot/resumo_gnuplot.pdf
http://www.uel.br/projetos/matessencial/superior/pdfs/Gnuplot-Ajuste.pdf
http://www.notasemcfd.com/2010/02/gnuplot-introducao-basica.html

Referências

Goto, K., and van de Geijn, R. A. 2008. Anatomy of High-Performance Matrix Multiplication, ACM Transactions on Mathematical Software 34, 3, Article 12.

(Note: explains the design decisions for the GotoBLAS dgemm implementation, which also apply to your code.)
Chellappa, S., Franchetti, F., and Püschel, M. 2008. How To Write Fast Numerical Code: A Small Introduction, Lecture Notes in Computer Science 5235, 196-259.
(Note: how to write C code for modern compilers and memory hierarchies, so that it runs fast. Recommended reading, especially for newcomers to code optimization.)
Bilmes, et al. The PHiPAC (Portable High Performance ANSI C) Page for BLAS3 Compatible Fast Matrix Matrix Multiply.
(Note: PHiPAC is a code-generating autotuner for matmul that started as a submission for this HW in a previous semester of CS267. Also see ATLAS; both are good examples if you are considering code generation strategies.)
Lam, M. S., Rothberg, E. E, and Wolf, M. E. 1991. The Cache Performance and Optimization of Blocked Algorithms, ASPLOS'91, 63-74. (Note: clearly explains cache blocking, supported by with performance models.)

Arquivos:
http://download.intel.com/support/processors/xeon/sb/xeon_E5-2600.pdf

Última atualização: quarta, 23 Ago 2017, 16:10